UDC: 004.8 https://doi.org/10.15407/jai2025.04.146

O. Rechynskyi[1], **B. Sokolovskii**[2], **O. Sinkevych**[3]
[1,2,3]Ivan Franko National University of Lviv, Ukraine
Dragomanov Street, 50, Lviv 79005
[1]oleksandr.rechynskyi@lnu.edu.ua
[2]bohdan.sokolovskyy@lnu.edu.ua
[3]oleh.sinkevych@lnu.edu.ua
[1]https://orcid.org/0009-0008-4427-6267
[2]https://orcid.org/0000-0002-2561-3255
[3]https://orcid.org/0000-0001-9834-2700

STUDY OF LOCAL PLANNING IN 2D PATHFINDING BASED ON SMALL LANGUAGE MODEL

Abstract. A paper studies the ability of compact large language models (LLMs) to perform local path planning using only limited, sensor-like observations. While recent work typically employs LLMs as high-level planners within hybrid systems leaving low-level navigation to classic pathfinding algorithms, considerably less attention has been given to how smaller models behave when placed directly in the control loop. To address this gap, we evaluate a 1.7-billion-parameter model (Qwen3-1.7B) as a local planner in a continuous two-dimensional environment. The agent navigates toward a target within a bounded rectangular map containing varying numbers of obstacles. At each step, a ring of collision-free candidate waypoints is generated, and made available to the model through a textual table containing indices, coordinates, and distances to the goal. Given few-shot demonstrations, the model selects a single candidate index to determine the next move. We report performance in terms of success rate, path length, and efficiency relative to the straight-line distance, and analyze the effect of hyperparameters tuned via Optuna. The results provide the first quantitative characterization of Qwen3-1.7B as a local path planner, highlighting both its capabilities and limitations. The findings motivate further exploration of richer observation formats, memory mechanisms, and comparisons with algorithmic baselines.

Keywords: large language models, pathfinding, navigation, local planning, machine learning.

1. Introduction

Large language models (LLMs) have recently moved beyond purely textual tasks and are increasingly used as high-level planners. When prompted appropriately, LLMs can transform abstract input information into executable actions or subgoals, effectively acting as task planners in simulated and real environments [1]. In robotics, this capability has been leveraged in systems such as SayCan [2], which combines an LLM with learned value functions to ground natural language commands into feasible robot behaviors, and in frameworks like Code as Policies [3], where language models generate robot policy code that encodes reasoning and control logic. These works suggest that LLMs can provide non-trivial planning and decision-making capabilities when integrated with appropriate perception and control modules via prompting or tools.

Path planning and navigation are particularly attractive domains for probing the limits of such abilities. Recent research has begun to explore LLM-enhanced navigation pipelines where the model interprets natural language instructions and cooperates with classical motion planners. For instance, the Dynamic Chain of Instruction-based Planning (DCIP) [4] integrates an LLM with 2D occupancy grid maps to decompose high-level language commands into structured navigation plans and constraints, improving safety and adaptability in dynamic environments. Another hybrid approach [5] uses LLM as a global planner to generate a high-level route and an A* search algorithm to perform detailed step-by-step pathfinding based on the proposed sequence from LLM. In vision-and-language navigation, systems like MapGPT [6] maintain an online, linguistically encoded topological map to support global exploration and adaptive path planning with GPT-based agents. Collectively, these results indicate that LLMs can meaningfully contribute to navigation and path planning when paired with suitable environment representations, prompting, and traditional algorithms.

However, most effort concentrates on either large proprietary or high-capacity open models or hybrid architectures where the LLM operates at a huge level, while the actual

pathfinding is delegated to classical planners (e.g., A*, D*). In many of these systems, the model has access to rich global context in the form of occupancy grids, topological maps, or symbolic descriptions of the environment, and its primary role is to interpret instructions, take into account semantic constraints, or re-rank candidate trajectories. By contrast, comparatively little work has systematically examined what small LLMs (1-7B parameter range) can do when placed in a low-level scenario and asked to make step-by-step navigation decisions based only on local sensor-like observations and goal information, without an explicit global map or a classical planner in the loop. These gaps should be addressed for the following reasons. First, in resource-constrained settings (e.g., embedded platforms), large LLMs are impractical, whereas small models might be deployable if they demonstrate sufficient competence in simple navigation tasks. Second, using purely local observations such as LIDAR-like range scans removes the privilege of known global representation and reveals more directly whether a model is able to understand geometric structure and make consistent local decisions. Lastly, pathfinding on simple 2D maps is a well-studied problem, making it a clean testbed for evaluating the limits of small LLMs.

In this paper, we empirically investigate the ability of small large language models (LLMs) to solve a basic two-dimensional (2D) pathfinding task within a LIDAR-like interaction loop. The environment is represented as a 2D map with no or only a small number of obstacles. At each step, the agent receives a set of observations and selects a movement direction that is expected to decrease the value of a given objective function. The model thus operates as a local planner, repeatedly mapping sensor-like observations and goal information to actions until the goal condition is satisfied. The obtained results help bridge the gap between high-level LLM-driven planning frameworks and purely algorithmic pathfinding methods, and provide a clearer picture of when, and to what extent, small LLMs can handle geometric navigation tasks.

2. Methods and means of implementations

2.1 Environment and map representation

We consider a continuous two-dimensional environment in which an agent must navigate from an initial position $s \in \mathbb{R}^2$ to a fixed goal position $g \in \mathbb{R}^2$. The environment is a bounded rectangular region

$$\Omega = [0, L_x] \times [0, L_y] \subset \mathbb{R}^2$$

The agent is modeled as a point moving in this plane, with the coordinates represented as (x, y) points. In the current setup, we evaluate the pipeline on maps with no or few obstacles:

- For maps without obstacles, the only constraints are the boundaries of Ω.
- On maps with obstacles, a finite set of polygonal obstacles is embedded in Ω. These obstacles are treated as impassable regions, and candidate moves that cross them are removed during planning.

At a discrete time step t, the agent is at position $x_t \in \Omega$. The algorithm starts from $x_0 = s$. The agent must reach the goal region

$$G = \{x \in \Omega \mid \|x - g\|_2 \leq r_{goal}\},$$

where $r_{goal} > 0$ is the tolerance radius. An episode terminates successfully when $x_t \in G$. If the number of steps exceeds a maximum number of steps T_{max} without reaching G, the episode is terminated and counted as a failure.

2.2 Ray-based scan and candidate generation

Rather than selecting arbitrary continuous actions, the planner operates on a finite set of candidate positions generated by LIDAR-like scan around the current agent position. These candidates lie on a circle of fixed radius around the agent and represent possible next waypoints.

The scan is characterized by the two main parameters:

- Scan radius r_s: fixed distance from the current position to each candidate (also equal to the movement step length).
- Angular resolution N: number of rays per scan.

For a given position x_t, the planner enumerates rays with evenly spaced angles

$$\theta_k = \frac{2\pi(k-1)}{N}, k = 1, \dots, N,$$

and computes candidate positions

$$p_t^{(k)} = x_t + r_s(cos\theta_k, sin\theta_k)$$

Each candidate position is rounded to a fixed number of decimal places for numerical stability in the prompt. Candidates lying outside the workspace bounds Ω are discarded. When obstacles are present, the planner also checks whether the straight-line segment from the current position to the candidate intersects any obstacle region. If the segment $[x_t, p_t^{(k)}]$ crosses an obstacle, the corresponding candidate is discarded and never shown to the language model. In other words, the candidate set at each step contains only positions that are both inside Ω and reachable by a collision-free straight-line move of length r_s.

For every remaining candidate, the planner evaluates the Euclidean distance from that candidate to the goal,

$$d_t^{(k)} = \left\| p_t^{(k)} - g \right\|_2,$$

rounded to three decimal places. The resulting set of candidates at time t therefore consists of:

- An index i.
- A position $p_t^{(i)}$.
- The associated scalar "distance-to-goal" value $d_t^{(i)}$.

All candidates are considered collision-free, and every actual move taken by the agent has the same length r_s; only the heading changes from step to step. Obstacles influence the decision process indirectly, through the removal of blocked candidates from this set.

2.3 Prompt-based decision making

At each step, the planner asks the LLM to choose which candidate the agent should move to. The LLM therefore acts as a policy that maps a textual description of the current state to an index over the candidate set. The numerical state is summarized in text as:

- The current point coordinates.
- The goal point coordinates.
- A table of candidates, where each row contains a candidate index, the candidate position, and the distance from that candidate to the goal.

An example of the candidate description format is shown in Listing 1.

1: move to $(x_1, y_1) \rightarrow dst_to_goal = d_1$
2: move to $(x_2, y_2) \rightarrow dst_to_goal = d_2$
N: move to $(x_N, y_N) \rightarrow dst_to_goal = d_N$

Listing 1: example of candidate description format

The candidates are listed in the same angular order as the scan procedure, starting from the positive x-axis and going counterclockwise. No explicit information about the workspace bounds or obstacles is included in the prompt. The LLM only observes current and goal coordinates and the per-candidate distances to the goal. History is cleared between steps in the default configuration, so each decision is made based solely on the most recent scan.

The planner uses a fixed instruction template, in which the LLM is told that it controls a robot moving in a continuous 2D plane and must select one candidate index that yields the smallest distance to the goal among the provided options. To guide the model, the prompt includes several few-shot demonstrations. Each demonstration presents:

- A current point.
- A goal point.
- A candidate table with distances.
- The correct answer: the index associated with the lowest distance to goal.

The live query with the data from the current time step is then appended as the final block of the prompt, with the same structure, but without the answer.

The LLM response is treated as arbitrary text from which an integer index is extracted. The planner scans the response for the first integer and interprets it as the chosen candidate index. If this index falls in the valid range of candidates at that step, the corresponding candidate $p_t^{(i)}$ is selected. If no valid integer is found, or the number is out of range, a simple fallback is used: the planner defaults to the first candidate in the list.

The planner doesn't verify whether the chosen candidate has the smallest distance-to-goal – it trusts the LLM's selection. In such a way, we can directly measure how well the model can pick the candidates with the smallest distance.

2.4 LLM and hyperparameter tuning

The primary model used in this work is a Qwen3-1.7B [7], which is a small language model containing 1.7 billion parameters. It is employed as a general-purpose language model with no fine-tuning on navigation data. The navigation behavior emerges from the prompt design and hyperparameter choices.

Each planner step corresponds to a single LLM generation call with a chat-style prompt. Empirically, the performance of the LLM-based planner depends on both environment-level parameters (scan radius, angular resolution, etc.) and generation-level parameters (sampling temperature, nucleus/top-p parameters, etc.). To systematically explore this space and identify well-performing configurations, we perform hyperparameter optimization using the TPE sampler [8] from Optuna. The optimization process is as follows:

- A validation set of maps and start-goal pairs is generated and kept fixed throughout tuning.
- A search space is defined over:
  - Scan radius r_s;
  - Angular resolution M;
  - Subset of generation parameters: temperature, top-p, top-k, sampling mode.
- For each trial, a configuration is sampled from this search space. The planner is initialized with these parameters and evaluated on the validation data.
- For each episode, an objective score is computed based on the resulting path:
  - If the goal is reached, the score is equal to the path length;
  - If the goal is not reached, a penalty term is added to the path length to reflect failure.
- The trial score is calculated as the mean over episodes.
- The hyperparameter study minimizes this score, effectively favoring configurations that yield short, successful paths and penalizing long or unsuccessful trajectories.

After a predefined number of trials, the configuration with the best validation score is selected. This configuration is then fixed and used in the main experiments reported in the Results and Discussion section.

3. Results and discussions

All experiments reported in this section use the Qwen3-1.7B model as the LLM planner, configured according to the method described in the Methods section. The environment is represented as a 2D rectangular workspace with start and goal points sampled uniformly within the bounds and constrained to be far enough from each other. Table 1 reports the configurations used for evaluation.

Table 1. Used evaluation configurations after hyperparameter tuning

Environment	No Obstacles	1 obstacle	5 obstacles
Angular Resolution	24	16	32
Scan Radius	5.0	5.0	5.0
do_sample	True	True	True
Top P	0.9	0.9	0.9
Top K	10	15	10
Temperature	0.1	0.1	0.1

The planner generates a finite set of candidates on a circle around the current position at each step, and queries the LLM to select one candidate index. The goal region is defined by a tolerance radius $r_{goal} = r_s$, and each episode is limited to a maximum number of steps $T_{max} = 100$. A run is considered successful if the final agent position lies within the goal tolerance before reaching this step limit. All metrics for tested configurations are computed over a fixed set of episodes $n = 5$.

We evaluate the results using the following metrics:

- Success ratio: fraction of episodes in which the final position lies within the goal tolerance r_{goal} before reaching the step limit T_{max};
- Path length: the sum of Euclidean distances between consecutive waypoints

$$L = \sum_{t=0}^{T-1} \|x_{t+1} - x_t\|_2$$

- Path efficiency: the ratio between the actual path length and the straight-line distance from start to goal

$$\eta = \frac{L}{\|g - s\|_2}$$

All figures in this section visualize the obtained trajectories. The start point is shown as a green triangle, the goal is a red star, and

the path is the orange polyline with dots marking intermediate waypoints. Grey rectangles represent the obstacles.

Table 2. Evaluation results of the proposed approach on different environments

Metric/Environment	No obstacles	1 obstacle		5 obstacles (map 1)	5 obstacles (map 2)
Success Ratio	100%	100 %		100%	80%
Path Length	27.69	27.24		46.15	36.90
Path Efficiency	1.148	1.089		1.871	1.532

Figure 1 shows an obtained trajectory in an obstacle-free environment. The agent starts in the upper left of the map and must reach a goal near the lower-right corner. The quantitative results are shown in Table 2, Row "No obstacles". Using the Qwen3-1.7B planner, the resulting path has a total length of 27.69 units. Because the environment contains no obstacles, the optimal path is simply the straight-line segment between start and goal. The obtained trajectory almost follows this straight line, deviating only due to the finite angular resolution of the candidate set and the reasoning ability of the LLM to select the best candidate at each step.

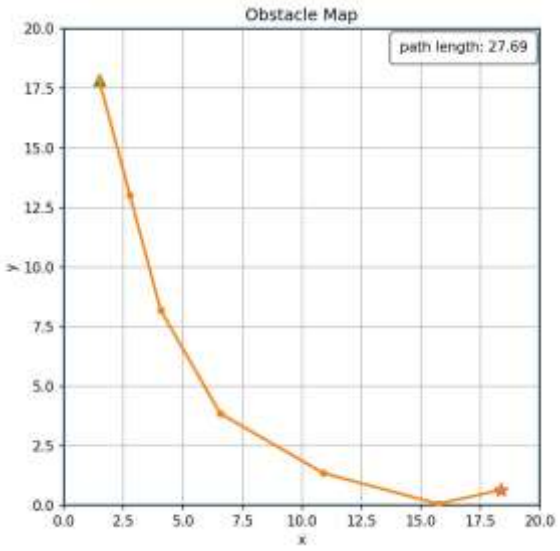

Fig. 1. The obtained path in an obstacle-free environment

To probe how robust this strategy is to mild clutter, we add a single rectangular obstacle between the start and the goal while keeping the same environment size. Figure 2 and Row "1 obstacle" in Table 2 show the obtained trajectory and metrics. The obstacle is placed near the center of the workspace, roughly along the direct line between start and goal, but leaving free passes on either side. The path produced by the planner has a total length of 27.24 units, and lies closer to the optimal trajectory line than in Figure 1. This can be explained by the discrete sampling effects, the exact candidate set configuration at the given time step, and the non-deterministic behavior of LLMs.

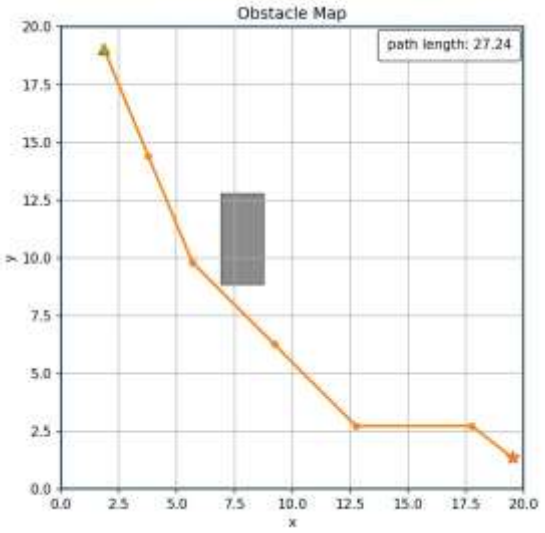

Fig. 2. The obtained path in an environment with 1 obstacle

From a geometric perspective, the planner behaves sensibly: it descends toward the central region, bends around the obstacle, and then continues with an almost straight segment to the goal. Such behavior is noteworthy since the planner does not explicitly encode the obstacle in the prompt. The only information given to the LLM consists of candidate positions and their distances to the goal; any candidates lying inside the obstacle are filtered out before the prompt is built. As a result, the model experiences the obstacle indirectly: it simply never sees certain candidate directions that would otherwise be attractive according to distance-to-goal.

Next, we consider more cluttered environments containing five obstacles each, again with the same overall workspace and a similar start-goal configuration. Figures 3 & 4 and Rows "5 obstacles (map 2)", "5 obstacles

(map 2)" in Table 2 reflect the result on five-obstacle setups.

In Figure 3, the agent eventually reached the goal, but only after a relatively long and indirect path. The total path length is 45.16 units with worse path efficiency than in the empty and single-obstacle cases. The trajectory reveals the planner's difficulty in more complex layouts. In the upper left quadrant, the agent initially moves downwards, but then performs several back-and-forth steps near clusters of obstacles. These oscillations occur because the LLM continues to choose candidates that appear favorable in terms of distance-to-goal but are, in fact, aligned with obstacles that the environment silently removes from the candidate set. The planner thus experiences a distorted local view of the distance field, and without memory or explicit obstacle information, it has no principled way to escape local valleys created by the obstacles.

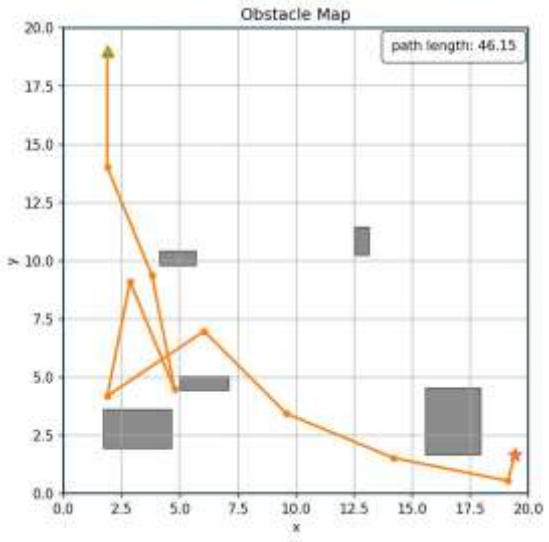

Fig. 3. The obtained path in an environment with 5 obstacles (map 1)

After several such exploratory steps, the agent finally discovers a viable corridor, after which the path straightens and becomes more goal-oriented. The second half of the path resembles clean trajectories seen in the simpler maps, but the earlier detours affect the total path length.

Figure 4 shows another configuration of a five-obstacle environment, where the agent again manages to reach the goal, this time with a total path length of 36.90 units. The resulting path is less noisy in the initial phase than in the previous path, but still shows detours. The agent first moves diagonally downwards and then makes a large S-shaped maneuver, avoiding obstacles that lie both above and below the direct start-goal line. Close to the right side of the map, the path bends again to pass through a narrow free region before turning toward the goal.

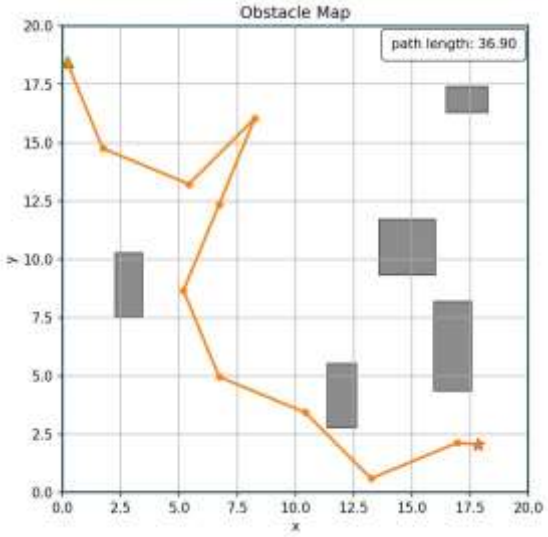

Fig. 4. The obtained path in an environment with 5 obstacles (map 2)

The trajectory demonstrates that the LLM-based planner can assemble a globally coherent route through complex arrangements, but in some cases it does so in not an optimal way: even though sharp oscillations are less frequent than in Figure 3, the cumulative deviation from the straight line is still large.

4. Conclusions

This paper evaluates the path planning capability of a small LLM, Qwen3-1.7B, used as a local controller in a continuous 2D navigation task. At each step, the agent receives collision-free candidate waypoints sampled within a circular neighborhood, described textually by indices and distances to the goal. The model selects a single candidate, forming a direct feedback loop where the LLM determines the next waypoint without task-specific training.

In obstacle-free maps, the planner consistently produces smooth, near-optimal trajectories with efficiency close to one. With a single obstacle, performance remains

comparable, as invalid waypoints are filtered and the agent implicitly navigates around the obstruction. In more cluttered environments, however, the model often enters loops and performs inefficient local search due to limited observation and lack of memory.

Hyperparameter tuning with Optuna shows that success and efficiency are highly sensitive to the model configuration. The results highlight both the promise and the limitations of small LLMs as local planners: they perform well in simple settings but require richer observations, memory mechanisms, or hybrid approaches to scale to more complex navigation tasks.

References

1. Huang, W., Abbeel, P., Pathak, D., & Mordatch, I. (2022). Language Models as Zero-Shot Planners: Extracting Actionable Knowledge for Embodied Agents. ArXiv (Cornell University). https://doi.org/10.48550/arxiv.2201.07207

2. Ahn, M., Brohan, A., Brown, N. A., Yevgen Chebotar, Andres, O., David, B., Finn, C., Gopalakrishnan, K., Hausman, K., Herzog, A., Daniel, Hsu, J., Ibarz, J., Ichter, B., Irpan, A., Jang, E. B., Ruano, R. J., Jeffrey, K., Jesmonth, S., & Joshi, N. J. (2022). Do As I Can, Not As I Say: Grounding Language in Robotic Affordances. https://doi.org/10.48550/arxiv.2204.01691

3. Liang, J., Huang, W., Xia, F., Xu, P., Hausman, K., Ichter, B., Florence, P., & Zeng, A. (2022). Code as Policies: Language Model Programs for Embodied Control. ArXiv (Cornell University). https://doi.org/10.48550/arxiv.2209.07753

4. Doma, P., Arab, A., & Xiao, X. (2024, December 3). LLM-Enhanced Path Planning: Safe and Efficient Autonomous Navigation with Instructional Inputs. https://doi.org/10.48550/arXiv.2412.02655

5. Meng, S., Wang, Y., Yang, C.-F., Peng, N., & Chang, K.-W. (2024). LLM-A*: Large Language Model Enhanced Incremental Heuristic Search on Path Planning. ArXiv (Cornell University). https://doi.org/10.48550/arxiv.2407.02511

6. Chen, J., Lin, B., Xu, R., Chai, Z., Liang, X., & Wong, K.-Y. (2024). MapGPT: Map-Guided Prompting with Adaptive Path Planning for Vision-and-Language Navigation. Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), 9796–9810. https://doi.org/10.18653/v1/2024.acl-long.529

7. Yang, A., Li, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Gao, C., Huang, C., Lv, C., Zheng, C., Liu, D., Zhou, F., Huang, F., Hu, F., Ge, H., Wei, H., Lin, H., Tang, J., & Yang, J. (2025). Qwen3 Technical Report. ArXiv (Cornell University). https://doi.org/10.48550/arxiv.2505.09388

8. Watanabe, S. (2023, May 26). Tree-Structured Parzen Estimator: Understanding Its Algorithm Components and Their Roles for Better Empirical Performance. ArXiv.org. https://doi.org/10.48550/arXiv.2304.11127

The article has been sent to the editors 30.11.25.
After processing 10.12.25.
Submitted for printing 30.12.25.

Copyright under license CCBY-SA4.0.